VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 Dominik Vašíček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Dominik Vašíček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: BiddingTools s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek, Ph.D.**

Konzultant bakalářské práce: Bc. Jiří Kostov

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 3. Května 2020 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 13. dubna 2020

Rád bych na tomto místě poděkoval Ing. Pavlu Dohnálkovi, Ph.D. za poskytnuté konzultace, které byly velmi nápomocné při zpracování této práce. Dále děkuji Bc. Jiřímu Kostovovi za podporu a všem členům týmu za přínosnou spolupráci při výkonu praxe.

Abstrakt

Tato práce popisuje průběh odborné praxe ve firmě Biddingtools Group s.r.o. V úvodu představuje zmíněnou firmu, tým, který stojí za produktem, jehož se tato práce týká, a úkoly, jenž jsou náplní této praxe.

Předmětem této praxe je vývoj jednoho z produktů, které tato firma nabízí. Tento produkt je editorem XML feedů se zaměřením pro marketingové specialisty. Hlavním cílem byl vývoj nových funkcí tohoto editoru s důrazem na návrh a testovatelnost nových součástí systému. Práce popisuje průběh zpracování čtyř úkolů na backendové části tohoto editoru. U každého úkolu je analyzován současný stav, navrženo možné řešení, uvedené detaily implementace a popsána revize řešení.

Práce je uzavřena pohledem na získané i využité znalosti během praxe a celkovým zhodnocením přínosu této praxe.

Klíčová slova: XML, vývoj webových aplikací, PHP, odborná praxe

Abstract

The present thesis describes the process of a professional practice in the company Biddingtools Group s.r.o. Introduction focuses on the mentioned company, team developing the product that is subject of this thesis and tasks that were done during this practice.

Aim of this practise is development of one of the products the company offers. The product is an XML feed editor with focus on marketing specialists. The main goal was to develop new features of this editor with emphasis on design and testability of created components. The thesis describes the progress of four tasks on the back-end part of this editor. For each task the current state is analyzed, a possible solution is designed, implementation details are presented and the solution is revised.

The thesis is concluded by a review of knowledge aquired and aplied during the practise and overall evaluation of its benefit.

Keywords: XML, web application development, PHP, professional practice

Obsah

Seznam použitých zkratek a symbolů **9**

Seznam obrázků **10**

Seznam výpisů zdrojového kódu **11**

1 Úvod **12**

2 Představení společnosti a produktu **13**
2.1 Společnost . . . 13
2.2 Produkt . . . 13
2.3 Tým a vývoj . . . 13

3 Využité technologie **15**
3.1 Jazyky . . . 15
3.2 Frameworky . . . 15
3.3 Knihovny a nástroje . . . 16
3.4 Standardizace . . . 16
3.5 Služby . . . 17

4 Podpora dynamických typů dat **18**
4.1 Obecné seznámení s Custom feedem . . . 18
4.2 Návrh implementace Custom feedu . . . 18
4.3 Implementace Custom feedu . . . 19
4.4 Revize implementace Custom feedu . . . 22

5 E-mailing systém **23**
5.1 Stav e-mailingu před implementací . . . 23
5.2 Návrh implementace e-mailngového systému . . . 23
5.3 Implementace e-mailingového systému . . . 23
5.4 Revize implementace e-mailingového systému . . . 29

6 Automatizovaný import manuálních změn **30**
6.1 Původní stav práce s manuálními změnami a jejich importem . . . 30
6.2 Návrh zjednodušení práce s manuálními změnami . . . 30
6.3 Implementace fronty pro import manuálních změn . . . 31
6.4 Revize úpravy importu manuálních změn . . . 32

7 Kontrola archivovaných feedů **33**
7.1 Stav podpory archivovaných feedů před implementací 33
7.2 Návrh implementace kontroly archivů 33
7.3 Implementace kontroly archivovaných feedů 34
7.4 Revize práce s archivovanými feedy 35

8 Závěr **36**
8.1 Znalosti získané studiem, které byly uplatněny v praxi 36
8.2 Chybějící znalosti získané praxí 36

Literatura **37**

Seznam použitých zkratek a symbolů

XML	–	Extensible Markup Language
PHP	–	PHP: Hypertext Preprocessor
API	–	Application Programming Interface
REST	–	Representational State Transfer
URL	–	Uniform Resource Locator
ORM	–	Object Relational Mapping
DOM	–	Document Object Model
UUID	–	Universally Unique Identifier
CSV	–	Comma Separated Values
DTO	–	Data Transfer Object
HTML	–	Hypertext Markup Language
FQN	–	Fully Qualified Name
SMTP	–	Simple Mail Transfer Protocol
MIME	–	Multipurpose Internet Mail Extensions

Seznam obrázků

1 Ukázka definice Custom feedu 21
2 Ukázka výsledné podoby e-mailové zprávy 25

Seznam výpisů zdrojového kódu

1 Příklad definice Google Atom feedu 22
2 Ukázková implementace celé e-mailové zprávy 27
3 Definice výjimky pro filestream wrapper 34

1 Úvod

Předmětem této práce je vývoj aplikace působící na poli e-commerce, která denně spravuje stovky produktových feedů e-shopů a dodavatelů zboží. Produktový feed (nebo jenom feed) je soubor obsahující všechna data o produktech a jejich atributech. Taková data mohou být například: název produktu, kategorizace, seznam parametrů, cena, jednoznačný identifikátor nebo textový popis produktu. Často jsou však doplněny dalšími informacemi specifickými pro daný typ feedu nebo jeho využití.

XML feedy jsou nedílnou součástí e-commerce a jsou většinově využity pro sdílení údajů týkajících se inzerce zboží na internetu. Produktové feedy jsou ze strany dodavatelů zboží však zřídka unifikovány, a proto je nutná transformace do konkrétních formátů dalších služeb vyskytujících se v e-commerce. Srovnávače zboží obecně přijímají pouze vlastní formáty feedů. Po e-shopech tedy vyžadována prezentace svého produktového portfolia v několika různých formátech, specifických pro každý srovnávací portál, na kterém chce své produkty nabízet. Mnohá e-shopová řešení také postrádají možnost editace svých produktových feedů, nebo je editace produktů časově nákladná při větším množství produktů. Tento editor XML feedů efektivně řeší všechny zmíněné problémy.

Firmou, produktem a týmem, který byly nedílnou součástí této praxe, se zabývá kapitola 2. Kapitola 3 seznamuje s nástroji a jazyky, které byly využity pro zpracování úkolů, a vysvětluje jejich význam v rámci daných řešení. Kapitola 4 popisuje vyřešení úkolu v rámci individuální praxe. Jejím předmětem je služba, která řeší problematiku různých definicí XML feedů pomocí transformace produktových feedů. Dalším zpracovaným úkolem se zabývá kapitola 5, která popisuje nahrazení současného řešení univerzálnějším a jednodušším na správu a monitoring. Třetím zpracovaným úkolem, řešeným v kapitole 6, je automatizace tvorby specifických úprav produktů, v editoru označovaných jako manuální změny. Kapitola 7 popisuje implementaci řešení, díky kterému je možná pokročilejší diagnostika archivovaných feedů a lepší informovanost poskytovatelů o chybě v archivu. Závěrečná kapitola 8 shrnuje průběh celé praxe z pohledu studenta, její přínos a využití znalostí získaných během studia.

2 Představení společnosti a produktu

Následující kapitola představuje společnost Biddingtools Group s.r.o., produkt, který je předmětem této individuální praxe, tým stojící za vývojem tohoto produktu a týmové metodiky, které provázely vývoj.

2.1 Společnost

Biddingtools Group s.r.o. [1] působí na trhu 6. rokem v oboru e-commerce. Cílem společnosti je optimalizace inzerce na zbožových porovnávačích a asistence e-shopům na online trhu. Společnost sídlí v centru Ostravy a aktuálně má přes dvě desítky zaměstnanců. Její portfolio služeb je poměrně široké, mimo jiné se mezi ně řadí například automatický bidding, párování produktů, generování cenotvoreb na míru, editace XML feedů včetně XML auditu pro jednotlivé srovnávače zboží nebo specifická technická řešení na míru. Hlavními předměty softwarového vývoje jsou zmíněné nástroje, které jsou vyvíjeny vnitropodnikovým IT oddělením. V roce 2020 se přidala ke skupině ette capital a.s.[2] a funguje pod novou obchodní společností trayto a.s. V současné době působí na mnoha trzích Evropy, s největším podílem v České a Slovenské republice.

2.2 Produkt

Editor XML feedů, jehož vývoj je předmětem této práce, je vyvíjen již 5 let a byl uveden na trh v roce 2017. Jeho hlavním úkolem je generování XML feedů dle dat obsažených ve vstupním feedu, zpracování uživatelem vytvořených pravidel nebo manuálních úprav pro změnu obsahu jednotlivých elementů produktů. Mezi hlavní přednosti tohoto editoru se řadí rychlost zpracování bez ohledu na velikost feedu, četnost aktualizací výstupního feedu, dostupnost generovaných feedů a podpora téměř všech formátů XML feedů.

Samotná aplikace se dělí na frontendovou část a backendovou část. Frontendová část je založena na PHP [3] frameworku Nette [4] a javascriptové knihovně jQuery. Tato část je rozhraním editoru, kterou dále využívají uživatelé pro editaci jejich feedů. Nevyužívá žádný databázový systém, jediným zdrojem dat je backendová část. Backendová část je rovněž psána v jazyce PHP a základem je framework Slim [5]. Backendová část poskytuje všechna potřebná data pro frontendovou část pomocí REST API. Dále využívá databázový systém Maria DB [6] a framework pro mapování databázových entit a práci s databází Doctrine [7]. Tato práce popisuje vývoj pouze backendové části tohoto editoru.

2.3 Tým a vývoj

Tým, který se podílel na vývoji aplikace během trvání praxe, měl proměnný počet členů. Mezi hlavní členy týmu se řadili 4 lidé, a to Backend vývojář, Frontend vývojář, Produktový manažer a Produktová specialistka. Další 2 Backend vývojáři byli primárně součástí jiných projektů s občasným zapojením do vývoje tohoto editoru.

Proces vývoje byl inspirován agilními metodikami vývoje. Pro sledování průběhu vývoje byl využit kanban, samotný vývoj probíhal ve formě iterací s proměnnou délkou, nejčastěji však týdenní iterace. Na definici jednotlivých uživatelských scénářů se podílel především Produktový manažer, všechny uživatelské scénáře byly kontrolovány týmem pro zajištění jasného cíle daného scénáře. Definované úkoly byly uloženy do backlogu a postupně odebírány vývojáři. Součástí každého zpracování úkolu byla jak předimplementační konzultace a návrh řešení, tak i revize kódu pro zajištění vhodného řešení daného uživatelského scénáře. Po úspěšné revizi kódu provedl Produktový manažer, případně i Produktová specialistka, revizi z uživatelského hlediska na jednom ze serverů určených pro kontrolu těchto řešení. Po schválení hotové úpravy se řešení přidalo do následující verze nahrané do produkce na konci iterace.

V rámci vývojářské části se dbalo na kvalitní spolupráci všech členů týmu. Každý den probíhaly pravidelné standupy, kde každý člen popsal výsledek své včerejší práce a stanovil své závazky na daný den. Vývoj byl inspirován Domain-driven designem [8]. Celá aplikace byla dělena na menší celky (domény). Celkově byla aplikace a její části jednoznačně popsány a pojmenovány tak, aby usnadnily komunikaci v celém týmu. Tým dbal na efektivní využití návrhových vzorů ve vhodných případech. V rámci modelové vrstvy byly využity entity, value objekty, továrny, repozitáře i služby – vše součásti Domain-driven designu.

Další součástí vývoje byla standardizace kódu pomocí různých nástrojů. Definované standardy byly sdíleny mezi oběma částmi aplikace a zajišťovaly stejnou podobu kódu všech programátorů. Dále odhalovaly neefektivní zápisy metod, možné chyby a typovou kontrolu. Všechna tato opatření zásadně zvýšila kvalitu a robustnost celého systému a umožnila rozsáhlejší refaktorování jednotlivých částí systému.

3 Využité technologie

Kapitola seznamuje s jádrem systému pohledem na využité jazyky a technologie v systému. Dále uvádí knihovny a nástroje, které byly využity v rámci plnění úkolů popsaných v této práci včetně nástrojů a knihoven třetích stran pro standardizaci kódu.

3.1 Jazyky

PHP

PHP [3] je interpretovaný skriptovací jazyk napsaný v C/C++, využíván převážně pro vývoj webových aplikací. Původně sloužil pouze pro vkládání dynamického obsahu do HTML dokumentů, později se však rozšířil na plnohodnotný objektově orientovaný programovací jazyk. Velkým milníkem pro tento jazyk je verze 7, která zavedla možnost využití striktního statického typování a výkonově mnohonásobně předčila předchozí verzi. PHP se těší velké popularitě na celém světě díky jeho jednoduchosti a dobré podpoře komunit.

XML

XML [9] je sada pravidel pro konstrukci značkovacích jazyků. Jeho zápis je jak strojově, tak lidsky čitelný. XML má své využití při přenosu dokumentů či strukturovaných dat. Editor používá existující XML feedy jako zdroj dat a generuje XML feedy nové.

3.2 Frameworky

Slim Framework

Slim [5] je open source PHP microframework využitý jako základ pro backend aplikace. Mezi jeho hlavní výhody patří rychlost, jednoduchost použití a kvalitní dokumentace. Úkolem tohoto frameworku je hlavně směrování HTTP požadavků a práce s nimi. Slim neobsahuje velké množství jiných komponent, proto je snadné s ním pracovat univerzálně.

Doctrine ORM

Doctrine ORM [7] je open source PHP framework určený pro propojení aplikace a databáze za pomocí mapování databázových entit. Je inspirován java frameworkem Hibernate. Doctrine také přináší jazyk DQL, který je vytvořený po vzoru jazyku HQL z Hibernate. Jeho značnou výhodou je možnost přenechání celé komunikace s databází na tomto frameworku, což značně usnadňuje a zefektivňuje jakoukoliv práci s databází. Framework využívá mnoho návrhových vzorů, pro mapování entit využívá mimo jiné *Data mapper* a *Lazy loading* a pro zápisy využívá například *Active record* nebo *Unit of work*. Další komponentou tohoto frameworku využitou v projektu jsou migrace, které obstarávají aktualizace databázového schématu včetně migrace dat a *data-fixtures*, jež jsou využity pro definici základních dat a testování aplikace.

Symfony

Aplikace využívá komponenty PHP frameworku Symfony [10], a to console a filesystem. Komponenta console je použita jako rozhraní pro ovládání aplikace vývojáři a umožňuje provádění operací mimo možnosti REST API aplikace. Komponenta filesystem je využita převážně pro práci s XML feedy nebo CSV soubory.

3.3 Knihovny a nástroje

Guzzle

Guzzle [11] je HTTP klient pro PHP. Tento klient umožňuje dotazování pomocí jednoduchého API. Guzzle umožňuje odesílat synchronní i asynchronní požadavky. V aplikaci je využit především pro stahování vstupních feedů nebo CSV souborů.

SwiftMailer

SwiftMailer [12] je univerzální knihovna pro práci s e-mailovými zprávami napsaná pro PHP, která je součástí frameworku Symfony. V editoru je využita v rámci implementace e-mailingového systému.

Twig

Twig [13] je populární šablonovací systém pro PHP. Mezi jeho přednosti se řadí rychlost, bezpečnost a flexibilita. Je součástí frameworku Symfony. Tento systém je v aplikaci využit pro generování obsahu e-mailových zpráv.

Git

Git [14] je verzovací systém využitý v obou částech aplikace. V projektech se dbá na kvalitní a čistou práci s gitem. Každá úprava v hlavní větvi je funkčním stavem aplikace a nese číslo zpracovaného úkolu a stručný popis daného úkolu.

3.4 Standardizace

PHPUnit

PHPUnit [15] je testovací framework pro PHP. pomocí tohoto frameworku jsou implementovány 3 typy testů. Prvními jsou Unit testy, které mají za úkol testovat jednotlivé části aplikace (často pouze jednotlivé třídy). Druhými testy jsou Smoke testy, které jednoduše volají koncové body aplikace a pouze ověřují, že je požadavek v pořádku, nebo je vyhodnocena chyba požadavku, přičemž nesmí nastat pád aplikace. Posledními testy jsou Integrační testy. Ty v aplikaci kontrolují správnou spolupráci komponent.

PHPStan

PHPStan [16] je open source nástroj pro PHP, který provádí statickou analýzu kódu. Statickou analýzou kódu je myšlena kontrola chyb bez spuštění kontrolovaného kódu. Mezi chyby, které PHP interpreter nekontroluje, a mohou být takto nalezeny, se řadí například kontrola existence tříd uvedených v některých konstrukcích, existence metod a funkcí, kontrola typů parametrů a návratových typů, kontrola existence proměnných a členských proměnných, kontrola podmínek apod. Tento nástroj společně s testy znatelně zvyšuje robustnost celé aplikace.

PHP Code sniffer

PHP Code sniffer [17] je jednoduchým nástrojem pro kontrolu konzistentnosti psaného kódu. Zajišťuje stejný styl kódu skrze celou aplikaci. Zároveň obsahuje možnost tyto chyby ve většině případů automaticky opravit.

PHP Mess detector

PHP Mess detector [18] je posledním z nástrojů určených pro kontrolu kvality kódu využitých v aplikaci. Jeho úkolem je vyhledávat neefektivně zapsaný kód, který by měl být opraven tak, aby byl čitelnější a srozumitelnější.

3.5 Služby

BitBucket Cloud

Bitbucket Cloud [19] je cloudová služba umožňující hostování Git repozitářů. Tato služba je využita pro všechny nástroje vyvíjené ve firmě BiddingTools. Nabízí přehledné rozhraní pro správu vzdálených repozitářů i pro revizi kódu pomocí pull requestů.

Circle CI

Circle CI [20] je cloudová služba zajišťující testování a nasazování verzí aplikace. Je propojena se službou BitBucket pomocí API. Díky tomu je testování a nasazování automatizované, jedinou uživatelsky provedenou akcí je nahrání nové verze do příslušné větve.

4 Podpora dynamických typů dat

Následující kapitola se zaobírá zpracováním úkolu Podpora dynamických typů dat. Tato funkcionalita je v projektu označována jako Custom feed a tento termín bude v celé práci využíván.

Kapitola prvně představí Custom feed a následně popíše proces zpracování úkolu od návrhu až po vydání do produkce.

4.1 Obecné seznámení s Custom feedem

Editor dosud zpracovával předem definované typy produktových feedů dle existujících specifikací daných srovnávačů zboží. Pro rozšíření spektra zpracovávaných produktových feedů se nabízela možnost definování formátu dalších feedů marketingovým specialistou velmi přínosnou, jelikož šetří čas programátorům a zároveň je výhodnou i z obchodního hlediska. Tato funkce byla marketingově nazvána Custom feed. Pomocí Custom feedu mohou být generované feedy využity nejen mezi e-shopy a srovnávači produktů, ale i dodavateli produktů a e-shopy.

4.2 Návrh implementace Custom feedu

Existující typy feedu (např. pro srovnávače Heureka.cz [21], Google Shopping [22]) poskytují tyto informace:

- název elementu ohraničující položky (Root element)
- název elementu ohraničující jednu položku (Item element)
- název elementu sloužící jako identifikátor položky
- název elementu sloužící jako název položky
- název elementu sloužící jako sekundární název položky
- název elementu určující kategorii položky
- název elementu určující URL adresu položky

Z těchto informací lze odvodit, že dosavadní definice typu feedu je svázána s produktovými feedy srovnávačů zboží. V případě definice produktového feedu dodavatele nemusí být některé položky využity. Hlavním úkolem je tedy rozšířit systém o možnost definice těchto produktových feedů skrze uživatelské rozhraní. Definované Custom feedy nebudou podporovat automatickou aktualizaci stromu kategorií, jelikož vyžadují specifické zpracování a často se odlišují formátem, strukturou a pravidly.

4.3 Implementace Custom feedu

Generalizace typu feedu

Entita *CustomFeed* nápadně připomíná entitu *FeedType*, jelikož mají v rámci aplikace sdílejí účel. Toto využití je tedy popsáno *Rozhraním typu feedu*, které vyžaduje od jeho implementací mimo výše definované parametry následující údaje:

- unikátní název typu feedu, pro standardní typy feedu jde o textový řetězec – u *CustomFeed* jde o UUID
- informaci o tom, zda se jedná o Custom feed či o standardní typ feedu
- šablonu typu feedu

Všechny současné reference mimo určité doménové entity jsou tedy nahrazeny za *Rozhraní typu feedu*, které je implementováno standardním typem feedu i *CustomFeed* typem.

Vytvoření doménové entity CustomFeedType

Entita *CustomFeedType* je další entitou mapovanou pomocí Doctrine ORM. Pro zjednodušení její definice využívá traitů pro identifikaci dle UUID, trait pro aktivovatelnost a trait pro sledování času vytvoření a upravení. Samotná definice entity popisuje práci s *Rozhraním typu feedu*. Dalším důležitým parametrem je *Eshop* (entita reprezentující konkrétní existující e-shop), ke kterému náleží daný *CustomFeed*, jelikož Custom feedy nejsou definovány pro všechny uživatele globálně, ale pouze pro *Eshop* daného uživatele.

Rozšíření doménové entity Feed o CustomFeedType

Doménová entita *Feed* je reprezentací produktového feedu, se kterým aplikace pracuje. Každý *Feed* náleží entitě *Eshop* a umožňuje načítat XML feedy jednoho typu a generovat XML feed druhého. Přidáním *CustomFeedType* se nyní rozšíří o další parametry, a to UUID vstupního a výstupního *CustomFeed*. V případě, že typ vstupního či výstupního feedu bude definován pomocí *CustomFeed*, jeho adekvátní atribut pro identifikátor standardního typu feedu bude nastaven na *custom* a bude povinně vyplněno UUID *CustomFeedType*, o který se jedná.

Důležitým aspektem je kontrola všech vložených dat, jak pro správné generování feedu, tak pro robustní doménovou logiku. Jelikož se jedná o převážně data, jejichž původ je uživatelský vstup, je třeba kontrolovat, zdali jsou názvy všech XML elementů platné. Pro tento účel je využita integrovaná knihovna DOM [23]. Jednoduchým pokusem vytvořit každý zadaný element je zjištěno, jedná-li se o platné názvy elementů. Dále je zajištěno to, že entita *Feed*, která definuje svůj typ vstupu jako *CustomFeedType*, má vždy existující implementaci. Je zakázáno mazat využité *CustomFeedType* z důvodu chybějící možnosti automaticky doplnit jakoukoliv obecnou definici takového feedu. Všechny tyto možné případy využití byly pokryty jak Unit testy, tak integračními testy.

Zpřístupnění funkcionality Custom feed Type pomocí API

Po zpracování doménové logiky a zajištění robustnosti dat v definované doméně byla funkcionalita zpřístupněna na API backendu. Byly vytvořeny následující koncové body s příslušnými oprávněními:

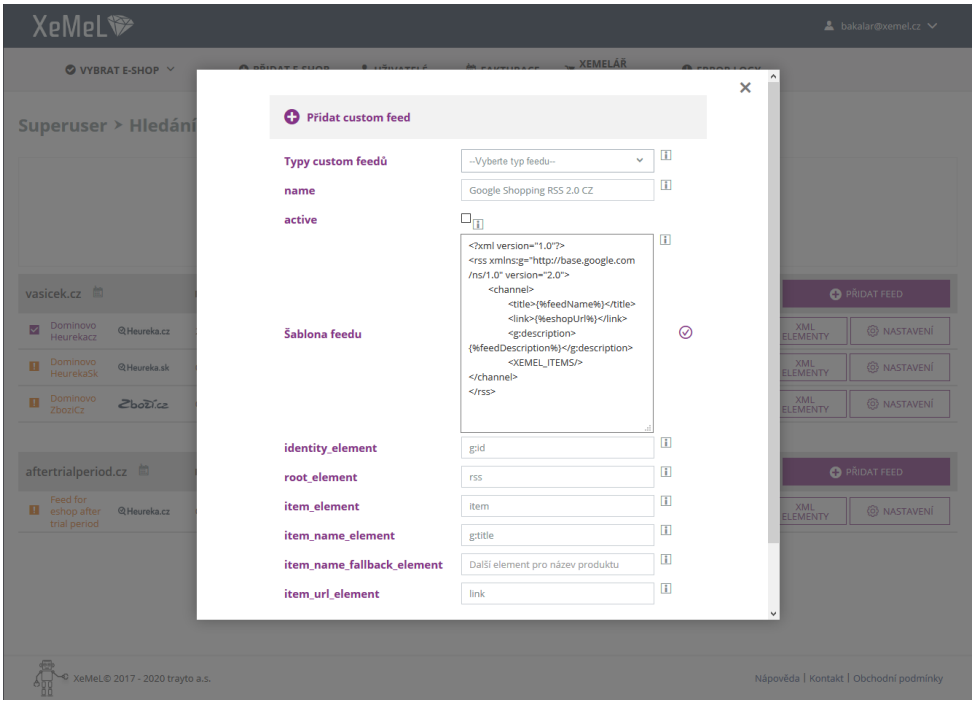
- získání všech definicí pro *Eshop* (Marketingový specialista)
- získání všech aktivovaných definicí pro *Eshop* (Správce *Eshopu*, Majitel *Eshopu*)
- získání dle UUID (Správce *Eshopu*, Majitel *Eshopu*)
- vytvoření definice (Marketingový specialista)
- aktualizace definice (Marketingový specialista)
- smazání definice (Marketingový specialista)

Rozšíření možnosti definice Custom feedu pomocí XML šablony

První uživatelský scénář požadoval pouze možnost definice feedu podobnou již stávajícím implementacím produktových feedu. Struktura těchto feedů je vždy podobná, jako první je deklarace XML dokumentu popisující verzi a kódování dokumentu, dále Root element, který obsahuje všechny Item elementy feedu, a Item elementy, které obsahují všechny parametry daných položek. Vzhledem k rozmanitým specifikacím jiných (především zahraničních) produktových feedů byl vytvořen druhý uživatelský scénář, který vyžadoval řešení, které pokryje širší spektrum existujících variant produktových feedů.

Toto řešení dále nabízí možnost dynamičtější práce s obsahem generovaného feedu. V rámci šablony můžeme označit určité místo, které se má vyplnit dynamickými daty. Dle některých specifikací feedů je například třeba uvést datum a čas, kdy byl feed aktualizován. Obsah tohoto, specifikací daného, pole můžeme tedy označit speciální proměnnou (v tomto případě *{%timeStamp%}*) a při generování feedu bude tato hodnota nahrazena aktuálním časem. Tato data mohou být doplněna informacemi z celého modelu aplikace. Mezi údaje, které jsou požadované některými specifikacemi feedů, se řadí například tyto:

- název feedu (*{%feedName%}*) – vyplněn dle jména feedu definovaným uživatelem v aplikaci
- název e-shopu (*{%eshopName%}*) – obvykle název klientova e-shopu
- url e-shopu (*{%eshopUrl%}*) – URL klientova e-shopu
- popis feedu (*{%feedDescription%}*) – název feedu + název e-shopu
- aktuální čas (*{%timeStamp%}*) – obvykle použit pro určení času poslední aktualizace



Obrázek 1: Ukázka definice Custom feedu

Editor pracuje především na úrovni samotných položek feedu, proto se řešení pomocí šablony definice feedu zdálo nejlepší. Šablona se skládá ze třech částí: ze začátku dokumentu (od deklarace XML dokumentu až po začátek Item elementů), z části s Item elementy a z části od konce Item elementů po konec dokumentu. Při samotném generování nového feedu se začátek a konec překopíruje do nového dokumentu a samotná část s položkami se zpracuje běžným způsobem. Ve finálním kroku se nahradí proměnné v šablonách za skutečné hodnoty.

Výhodou tohoto řešení je dobrá testovatelnost, protože lze testovat správné generování bez jakéhokoliv obsahu. Testování může simulovat vstupní feed bez obsahu a pouze transformovat prázdný obsah do nové šablony Custom feedu. Takto je zaručeno, že v případě selhání testu jde o chybu sestavení šablony a ne o chybu některé z komponent upravující obsah výstupního feedu. Testy mohou být zároveň vytvořeny na míru každému již definovanému typu feedu, jelikož jsou v rámci úpravy znovu definovány šablonou.

Posledním krokem v rámci úpravy je definice dvou *CustomFeedType* jako demodata aplikace. Tato demodata jsou používána pro integrační testování a následnou kontrolu funkcionality týmem. Využita byla ve dvou případech. Prvním případem byl pouze test generování, XML feed nebyl nijak změněn a svůj vstupní i výstupní typ definoval stejným Custom feedem. Druhý případ transformoval pomocí Custom feedu existující produktový XML feed do formátu definovaného jiným srovnávačem produktů.

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:g="http://base.google.com/ns
    /1.0">
   <title>{%feedName%}</title>
   <link href="{%eshopUrl%}" rel="alternate" type="text/html"/>
   <updated>{%timeStamp%}</updated>
   <author>
      <name>{%eshopName%}</name>
   </author>
   <TEMPLATE_ITEM_TAG/>
</feed>
```

Výpis 1: Příklad definice Google Atom feedu

4.4 Revize implementace Custom feedu

Proces vývoje podpory cizích produktových feedů začal prvotním návrhem implementace a integrace do struktury aplikace a byl ukončen nasazením rozšíření o definici Custom feedu pomocí XML šablony feedu. Byly tedy zpracovány dva uživatelské scénáře: definice Custom feedů pomocí formuláře a následná úprava definice pomocí XML šablon.

Definice feedů pomocí XML šablon nakonec nahradila původní komponentu pro generování feedu. Stávající definice typů feedů byly migrovány tak, aby byly definovány pomocí těchto XML šablon. Velkou výhodou je, že toto řešení pokryje většinu specifikací produktových feedů, které uchovávají všechny parametry jednotlivých položek seskupené pod jedním Item elementem. Pro feedy, které pracují s produkty tak, že seskupí parametry produktů a samotné produkty definují např. pomocí referencí na tyto hodnoty, toto řešení není vyhovující. Zpracování takovýchto feedů by bylo možné, pokud by se jako první první transformovaly do podoby položek se všemi parametry, ovšem poměr těchto feedů je malý, a tak je toto řešení přijatelné.

Testovatelnost tohoto řešení je zajištěna implementací dvou způsobů zapisování produktů. Jeden vytváří XML soubory a je využit v běžném procesu generování, druhý zapisuje pouze do paměti a další testy jsou tedy prováděny nad ním. Celé řešení je pokryto Unit testy, které testují komponenty jak pro zápis, tak komponenty pro nahrazování proměnných. Celý proces zápisu je pak kontrolován integračními testy.

5 E-mailing systém

Následující sekce popisuje kompletní refaktorování všech odesílaných e-mailů z aplikace. Úkol byl zpracován od prvotního návrhu až po nasazení do produkčního prostředí.

5.1 Stav e-mailingu před implementací

Aplikace využívala e-mailové komunikace nejen pro potvrzení registrace, informování uživatelů, ale i pro informování marketingových specialistů, vývojářů editoru o proběhlých událostech a stavu aplikace. Všechny definice těchto e-mailových zpráv byly definovány v rámci komponent, které měly svou vlastní odpovědnost a zároveň nesly odpovědnost za tvorbu a odeslání e-mailových zpráv. Jelikož e-mailové zprávy nebyly osamostatněny a vytvořeny na stejném základu se stejnou nebo obdobnou implementací, bylo potřeba tyto zprávy nějak sjednotit a zajistit jejich nezávislost na komponentách v rámci udržitelnosti do budoucna. Jejich následná testovatelnost by byla také značným přínosem. Pro tvorbu a odesílání se využívala knihovna Swift Mailer, univerzální knihovna pro práci s e-mailovými zprávami. Vzhledem k dobré zkušenosti s touto knihovou byla znovu využita v nové implementaci e-mailového systému.

5.2 Návrh implementace e-mailngového systému

Hlavním úkolem je vytvořit systém pro tvorbu a odesílání e-mailových zpráv tak, aby byl časově nenáročný jak pro tvorbu nových, tak pro editaci stávajících zpráv. Každá zpráva by také měla být testovatelná, aby bylo zajištěno správné fungování aplikace. Posledním důležitým požadavkem je nezávislost na komponentách, které budou s e-mailovými zprávami dále pracovat.

Pro zajištění jednoduché údržby a rozšiřitelnosti se nabízí řešení vyčlenění každé jednotlivé zprávy do příslušné třídy a delegovat všechny specifické požadavky pro danou zprávu takové třídě. Tímto zajišťujeme nezávislost od jiných komponent nebo služeb, které budou tyto zprávy v budoucnu využívat. Vzhledem k faktu, že většina aplikační logiky ohledně tvorby, zpracování a odesílání je téměř totožná pro všechny odeslané zprávy, testování těchto samostatných zpráv pouze pomocí Unit testů není nejoptimálnější. Samotné třídy reprezentující e-mailové zprávy nemají vlastní logiku a jedná se výhradně o nosiče dat pro služby, které se zprávami dále pracují. Vhodnější typ testů pro tyto zprávy jsou tedy integrační testy, které mají za úkol testovat komplexnější úlohy, které využívají více komponent najednou.

5.3 Implementace e-mailingového systému

Základem samotných e-mailových zpráv není abstraktní třída, sloužící jako rodič všech e-mailových zpráv, ale jsou to tři rozhraní. Rozhraní jsou záměrně rozdělena dle položky I - Interface segregation v SOLID principle [24]. Takto umožňujeme využít i jiné objekty jako zdroj dat pro budoucí e-mailové zprávy, případně můžeme vytvořit objekty reprezentující konkrétní šablony

s jejich požadavky na data, aniž bychom museli řešit kolizi mezi existující povinnou implementací v rodičovské třídě.

Rozhraní vykreslitené zprávy po třídách, které jej implementují, vyžaduje poskytování cesty k souboru se šablonou, která je určena pro danou zprávu a mapu hodnot, které slouží jako dynamické data dosazené do šablony e-mailu. E-mailové zprávy jsou vykreslovány pomocí šablonového systému Twig. Twig je šablonovým systémem pro framework Symfony, který se řadí mezi nejčastěji používané PHP frameworky. Jeho výhodou je možnost generovat mimo HTML i další typy textových dokumentů.

Druhé rozhraní je *Rozhraní e-mailové zprávy*, které po jeho implementacích vyžaduje údaje nutné pro odeslání e-mailové zprávy z hlediska aplikace. Všechny třídy implementující toto rozhraní musí definovat název dané zprávy, který je ve většině případu shodný s FQN dané třídy (jmenný prostor + název třídy). Dále třídy poskytnou informaci o tom, jestli se má daný typ zpráv v dané konfiguraci a sestavení odesílat. Poslední a nejdůležitější úkol je předání třídy *MessageData*, která nese informace specifické k e-mailu:

- seznam všech adresátů e-mailu
- předmět zprávy
- vygenerované HTML tělo zprávy
- odesílatel e-mailové zprávy
- přílohy e-mailové zprávy
- seznam e-mailů v kopii zprávy

Poslední rozhraní je *Rozhraní pro uložitelné zprávy*. Zprávy, u kterých chceme vytvářet záznamy v databázovém logu, implementují toto rozhraní. Zprávy mohou pak dle sebe vytvořit novou entitu *EmailMessage*, která je spravována a mapována pomocí frameworku Doctrine ORM.

Entita *EmailMessage* je jednoznačně identifikována skrze UUID generovaným ve využitém traitu. Dále v sobě uchovává informace o tom, kterému uživateli byla zaslána a jaká třída tuto zprávu reprezentuje. Posledními atributy jsou čas vytvoření a poslední úpravy, také pocházející z příslušného traitu.

Samotné sestavení těla e-mailu má za úkol služba *EmailRender*. V rámci podpory i stávajících e-mailových zpráv a možností postupného refaktorování existujících zpráv, má třída dvě hlavní metody, a to pro poskytnutí těla zprávy dle cesty k šabloně a dynamických dat zprávy a dále poskytnutí těla zprávy dle jakékoliv třídy implementující *Rozhraní vykreslitené zprávy*. Tato služba také do všech zpráv dosazuje všechny přílohy obsažené ve všech e-mailech (logo editoru, loga sociálních sítí atp.).

Obrázek 2: Ukázka výsledné podoby e-mailové zprávy

Továrna SwiftMessage zapouzdřuje práci s knihovou SwiftMailer tak, že nabízí jednoduché metody pro tvorbu Swift zpráv (viz. knihovna Swift message). Všechny zprávy vytvořené touto třídou jsou připraveny na finální odeslání pomocí služby pro odesílání zpráv. Celá továrna je dobře testovatelná Unit testy, protože vytvořené Swift zprávy poskytují vhodné API pro kontrolu zpráv.

Další podstatnou součástí e-mailového systému jsou správci zpráv (Message managers). Tito správci definovaným způsobem manipulují se zprávami dle aktuální konfigurace. Všichni správci, kteří chtějí při odeslání zprávy provést nějakou činnost s odesílanou zprávou, musí implementovat rozhraní pro správce zpráv.

Mezi takto implementované správce zpráv aktuálně patří:

- SMTP message manager – správce, který vytvoří Swift zprávu dle dat dostupných v objektu dané zprávy, doplní ji o informace vztažené k dané konfiguraci a odešle ji pomocí SwiftMailer maileru.

- Database message manager – správce, který všechny zprávy, implementující rozhraní pro uložitelné zprávy převede na mapované databázové entity *EmailMessage* a uloží je do databáze.

- Log message manager – správce, který pro každou odeslanou zprávu vytvoří záznam v log souboru ve souborovém systému aplikace. Tento log se používá převážně jako sekundární, protože uchovává záznamy o všech zprávách, které jsou poslány z aplikace včetně těch, které neimplementují rozhraní pro uložitelné zprávy.

- Temporary message manager – tento správce slouží hlavně pro testování celého e-mailingového systému tak, že uchovává všechny Třídy zpráv ve stavu, v jakém se měly odeslat. Tato úschova je dočasná, protože ukládá všechny zprávy v paměti a nikam tyto zprávy trvale neukládá.

- Chained message manager – tento správce sám o sobě nijak nemanipuluje se samotnou zprávou, jedinou funkci, kterou zajišťuje, je předání převzaté zprávy všem správcům zaregistrovaným v tomto správci. Jeho hlavní využití je tedy vytvoření jakéhosi řetězu, kterému bude postupně předána zpráva. Celá tato konstrukce je jednoduše zpřístupněna všem službám, které si o ni zažádají skrze *Dependency injection*.

Takto na míru sestavená kombinace správců zpráv může být velmi jednoduše a pohodlně vložena do všech objektů, které jsou vytvářeny pomocí dependency injection, protože jsou, až na Temporary message manager, bezstavové. Container je nedílnou součástí vzoru *Dependency injection*. Jeho cílem je uchovávat již existující instance tak, aby v případě opětovné nutnosti využít danou třídu, byla předána již existující instance. Dle údaje o aktuálním prostředí a konfiguraci se pod rozhraní pro správce zpráv v containeru vytvoří adekvátní objekt. Je-li aplikace v testovacím režimu, globálně dostupný message manager bude Temporary message manager. V opačném případě bude dostupná kombinace správců spojených pod Chained message manager vyhovující aktuálním požadavkům (např.: SMTP Message manager > Database Message manager > Log message manager).

Testování e-mailového systému probíhá na úrovni doménové (modelové) vrstvy, protože samotné Unit testy tříd e-mailů negarantují správnou funkčnost systému jako celku. Integrační testování v tomto případě simuluje události vyvolané uživateli či automatizovanými procesy a ověřuje nejen správné chování za modelovou vrstvou, ale zároveň pomocí Temporary message manageru kontroluje, jsou-li správně odesílány e-mailové zprávy. Pro zajištění nezávislosti testů je potřeba Temporary message manager po každém testu vyprázdnit, protože při testování se container nesestavuje znovu, ale používá se již existující. Pro tento případ se používá metoda pro vyprázdnění, která je spuštěna při uzavření každého testu.

Poslední součástí e-mailového systému jsou specifické služby pro vytváření e-mailových zpráv orientované na jednu konkrétní část domény. Tyto služby jsou vytvořeny dle návrhového vzoru *Factory* a jejich účel je poskytnout jednoduché a srozumitelné API pro vytvoření a odeslání

e-mailových zpráv. Pro jejich využití je potřeba jen minimum parametrů (vetšinou jedinná specifická entita nesoucí dynamická data e-mailové zprávy) a jsou snadno využitelné skrze celou modelovou vrstvu.

```php
<?php declare(strict_types=1);

namespace Core\Model\User\Email;

use Core\Components\Mailer\EmailRenderService;
use Core\Components\Mailer\MailerConfig;
use Core\Components\Mailer\MessageData;
use Core\Components\Mailer\MessageInterface;
use Core\Components\Mailer\RenderableMessageInterface;
use Core\Model\EmailMessage\EmailMessage;
use Core\Model\EmailMessage\StorableMessageInterface;
use Core\Model\User\User;

class WelcomeEmailMessage implements RenderableMessageInterface,
    MessageInterface, StorableMessageInterface {

   /** @var \Core\Model\User\User */
   private $user;
   /** @var \Core\Components\Mailer\EmailRenderService */
   private $emailRenderService;
   /** @var \Core\Components\Mailer\MailerConfig */
   private $mailerConfig;

   public function __construct(
      User $user,
      EmailRenderService $emailRenderService,
      MailerConfig $mailerConfig
   ) {
      $this->user = $user;
      $this->emailRenderService = $emailRenderService;
      $this->mailerConfig = $mailerConfig;
   }

   public function getUser(): User {
      return $this->user;
   }
```

```
    public function getTemplateData(): array {
        return [
            'email' => $this->user->getEmail()->asString()
        ];
    }

    public function getTemplatePath(): string {
        return 'user/welcomeEmail.html.twig';
    }

    public function getMessageData(): MessageData {
        return MessageData::createSimpleMessageWithInlineAttachments(
            [$this->user->getEmail()->asString()],
            'Dobrá práce, účet máte aktivní',
            $this->emailRenderService->renderByRenderableMessage($this),
            $this->mailerConfig->getDefaultMailSender(),
            $this->emailRenderService->getDefaultEmailInlineImageAttachments()
        );
    }

    public function getMessageName(): string {
        return self::class;
    }

    public function isEnabled(): bool {
        return $this->mailerConfig->isRegistrationNotificationEnabled();
    }

    public function toStorableEmailMessage(): EmailMessage {
        return new EmailMessage($this->user, $this->getMessageName());
    }
}
```

Výpis 2: Ukázková implementace celé e-mailové zprávy

5.4 Revize implementace e-mailingového systému

Proces vývoje e-mailingového systému sahá od prvotního návrhu až k nasazení do produkčního prostředí. Výše popsané řešení nese množství výhod. První výhodou je téměř neomezující návrh z pohledu rozšířitelnosti e-mailového sytému. Pro využití daného systému z pohledu tvorby zprávy je potřeba implementovat pouze jedno rozhraní. Objektům, které toto rozhraní mohou implementovat, se žádné meze nekladou. V případě plnění specifikace úkolu dle běžného uživatelského scénáře je definice e-mailové zprávy velmi jednoduchá a intuitivní i pro uživatele s nízkou znalostí systému, protože všechny požadavky povětšinou zpracovává třída daného e-mailu a není potřeba zasahovat do dalších částí systému. Pro ještě větší zjednodušení stačí tvorbu instancí této třídy delegovat do jedné ze služeb pro tvorbu e-mailů, kde požadavky na vytvoření e-mailové zprávy dosáhnou nutného minima.

Součástí této úpravy bylo i demonstrační refaktorování několika již existujících zpráv. Tento proces názorně ukázal, jak s tímto systémem dále pracovat a jak jej případně dále rozvíjet. Výsledkem takového refaktorování byly čtyři třídy zpráv, každá nahrazující existující implementaci, a dvě továrny pro snadnou tvorbu těchto zpráv. Nově vytvořené zprávy demonstrovaly proces extrakce existující logiky tvorby a odesílání zpráv do oddělených služeb a zjednodušení doménové logiky akcí, které e-maily odesílaly.

Další rozvoj je podpořen i návrhem správců zpráv. V případě požadavku na úpravu jednoho z nich nevzniká riziko ovlivnění jiných správců. Registrace dalších takových správců je velmi snadná a po implementaci rozhraní pro správce zpráv nabízí další možnosti práce s těmito zprávami. Jako příklad lze uvést notifikační systém, který bude o těchto zprávách notifikovat uživatele či produktový tým skrze nové nebo existující API, a nebo službu pro notifikaci uživatelů pomocí SMS.

6 Automatizovaný import manuálních změn

Následující kapitola popisuje celý proces vývoje, který umožnil aplikacím třetích stran provádět editace dat v editoru pomocí CSV souborů. Tento import byl implementován a nasazen do produkčního prostředí s tím, že návrh i testování bylo provedeno v rámci spolupráce s jiným týmem.

Manuální změnou je v doméně editoru pojmem označující jednoduchou úpravu obsahu jednoho produktu feedu. Manuální změna vždy nese informaci, o jaký konkrétní produkt se jedná (jeho identifikátor) a jaké údaje (hodnoty elementů) má pozměnit. Manuální změna tedy může přepsat například název konkrétního produktu nebo jeho kategorizaci.

6.1 Původní stav práce s manuálními změnami a jejich importem

Způsoby, kterými lze v editoru vytvořit manuální změnu, byly dva. Prvním byla změna produktu v rozhraní aplikace. Ta nabídla uživateli formulář obsahující seznam všech elementů daného produktu a uživatel mohl jednoduše přepsat tyto hodnoty. Výhodou tohoto způsobu byl dobrý přehled o původním i novém stavu produktu. V případě, že bylo potřeba editovat větší množství produktů (často tisíce produktů), toto řešení nebylo ideální z důvodu časové náročnosti editace. Jako druhý způsob se nabízel import těchto manuálních změn pomocí CSV souboru v administraci editoru. I ten však nebyl nejlepším řešením, jelikož měl určité limitace. Přijímal CSV soubory pouze kódované pomocí *Windows-1250* oddělené středníkem, v případě již existující změny pro daný produkt import selhal a navrátil manuální změny do původního stavu. Na základě uživatelské odezvy byl tedy vytvořen požadavek na zjednodušení tvorby velkého množství manuálních změn.

6.2 Návrh zjednodušení práce s manuálními změnami

Z definice požadavků vyplývá, že se bude úprava týkat převážně práce s dávkovým importem manuálních změn. Uživatele můžeme rozdělit do dvou skupin, a to na jeden z týmů, který import využívá pro provedení změn generovaných jinou aplikací a ostatní uživatele.

Pro maximální optimalizaci importu změn druhého týmu se nabízí vytvoření speciálního koncového bodu pro zaregistrování nového importu. Ten vzhledem k faktu, že jeho výsledek je druhým týmem analyzován nejdříve následující den, může být odložen na dobu, kdy systém není vytížen. Toto odložení nutí informovat tým o stavu importu jiným způsobem, než v odpovědi původního požadavku. Pro to může být využit e-mailingový systém popsaný v předchozí kapitole. Dalším požadavkem je možnost mazat existující manuální změny pomocí tohoto importu. Pro tento účel lze navrhnout speciální token, pomocí něhož lze odlišit běžnou aktualizaci od smazání změny.

V importu využívaném běžnými uživateli je potřeba rozšířit možnosti přijímaných CSV souborů. Pro obě skupiny uživatelů se pozmění logika importu změny v případě, že již pro daný

produkt existuje manuální změna. Takováto situace místo selhání vyústí v aktualizaci konkrétní manuální změny.

6.3 Implementace fronty pro import manuálních změn

Prvním dílčím úkolem je vytvořit frontu importů v modelové vrstvě aplikace. Celá tato funkcionalita je označena jako *Manual Change Import Task* (Dále *Import Task*). Třída vytvořena dle vzoru Domain Model zajistí uložení importů do databáze. Pro uložení do databáze je vytvořena adekvátní migrace databáze, vytvářející tabulku a vazby na ostatní entity svázané s tímto importem. Tato doménová třída eviduje následující údaje:

- generované UUID registrovaného importu
- vstupní feed, ke kterému se import bude vázat
- skupinu manuálních změn, do které budou změny zařazeny
- stav importu nabývající hodnot *Nový*, *Stažený* nebo *Dokončený*
- výsledek importu, který může být *Čekající*, *Úspěšný* nebo *Neúspěšný*
- čas, kdy byl import ukončen nehledě na výsledek importu
- URL adresa veřejně dostupného souboru, který bude použit pro import
- příznak, určující, jestli je stažený soubor v souborovém systému aplikace
- absolutní cestu k souboru, pokud je v souborovém systému
- kódování CSV souboru, oddělovač atributů a znak použitý pro nový řádek

Pro práci s tímto Domain Modelem je vytvořena DTO třída *ImportTaskData*, která nese všechny informace nutné pro tvorbu této třídy. Třída *ImportTaskRepository* nabízí metody pro vyhledání entit *ImportTask*, které ještě nebyly zpracovány, a pro vyhledání zpracovaných entit *ImportTask*, kterým mohly být smazány CSV soubory se změnami. Třída *ImportTaskFileRepository* obstarává mazání souborů ze souborového systému aplikace. Další modelovou třídou, vytvořenou dle vzoru *Facade*, je *ImportTaskFacade*. Ta, jakožto nejsvrchnější vrstvou modelu aplikace, nabízí mimo metody poskytnuté repozitářem i metody pro samotné zpracování importu a registraci importu z controlleru aplikace. Poslední a nejdůležitější třídou je služba provádějící samotný import. Ta ve své jediné veřejné metodě přijímá parametry, které poskytuje jak *ImportTask*, tak původní požadavek na import z API, který využívá frontend aplikace.

Jelikož je samotný účel obou importů stejný, zpracuje jej tato univerzální služba. Ta pomocí již existující služby pro čtení CSV souborů poskytující páry Identifikátor produktu -> Změněná data vytváří možné manuální změny (Instance DTO třídy). Má-li tento objekt v datech speciální token pro smazání existující manuální změny, pokusí se pomocí identifikátoru z CSV souboru

vyhledat existující manuální změnu a pak ji smaže. Pokud token neobsahuje, využije zmíněný repozitář pro vyhledání změny a v případě, že změna pro daný identifikátor existuje, bude aktualizována. V opačném případě proběhne vytvoření nové manuální změny.

Dalším zpracovaným požadavkem je informování uživatelů o výsledku odložených importů skrze e-mailingový systém. Ten odesílá dva druhy zpráv, a to informaci o úspěšném nebo neúspěšném importu. Pro každý je vytvořena samostatná třída obsahující informace o registrovaném importu, jeho výsledku a případný text zprávy při selhání importu. Mezi příčiny selhání patří například špatně uvedené kódování, oddělovač, konec řádku nebo nedostupný CSV soubor. Do konfigurace jsou přidány seznamy e-mailových adres pro obě e-mailové zprávy. Pro marketingový nebo optimalizační tým jsou důležité obě zprávy, pro vývojáře většinou jen ty chybové.

Automatické spouštění těchto importů obstarává třída implementující rozhraní pro úkoly spouštěné periodicky pomocí cronu. Ta vždy v čas, kdy se očekává nízká zátěž systému, vyhledá importy, které ještě neproběhly a předá je v modelu skrze fasádu pro zpracování import tasků. Druhá třída implementující rozhraní pro úkoly spouštěné periodicky pomocí cronu obstarává mazání CSV souborů úspěšně zpracovaných import tasků. Takto je zajištěn celý cyklus zpracování importů na modelové vrstvě.

Poslední částí je umožnění samotné registrace takového importu. Pro tento účel je vytvořen speciální koncový bod v API backendu. Aby tato funkcionalita zůstala omezená pro vybrané interní nástroje firmy, ověřuje pomocí služby omezující přístup k jednotlivým koncovým bodům aplikační token aplikace, která vznesla požadavek. Pro tento daný koncový bod je povolen přístup dvěma aplikacím, a to aplikaci Jack Párovač a Integračním testům, které také využívají API aplikace. Pro umožnění komunikace s Jackem Párovačem je vytvořena druhá migrace, přidávající do databáze jeho nový aplikační token.

Pro celou tuto funkcionalitu je vytvořena řada testů pokrývající nové scénáře. Je kontrolován celý proces importu a všechny scénáře, které mohou nastat v závislosti na obsah CSV souboru. Testy ověřují i správné odesílání e-mailových zpráv čtením *Temporary message manageru*. Dále je testováno stahování CSV souborů a jejich otevření pomocí definovaných údajů v registračním požadavku. Poslední sada testů se zabývá přístupem k vytvořenému koncovému bodu s použitím povoleného a nepovoleného aplikačního tokenu.

6.4 Revize úpravy importu manuálních změn

Podpora automatických importů byla společně s druhým týmem navržena, na straně aplikace zpracována, a po společném testu nasazena do produkčního prostředí. Samotné využití této funkcionality pro jakoukoliv další službu je velmi snadné, jelikož se jedná pouze o jeden koncový bod, který je podrobně zdokumentován. Dostatečnou informovanost zajišťují e-mailové zprávy odesílané po importu. Z frontendové strany koncového bodu byl vytvořen nový uživatelský scénář požadující možnost odeslat CSV soubory v jakémkoliv kódování, odděleno různými znaky. Celé řešení je důsledně testováno jak Unit, tak integračními testy.

7 Kontrola archivovaných feedů

Následující kapitola popisuje nedostatky současné práce s archivovanými feedy, důvody pro rozšíření kontroly archivovaných feedů o dva možné případy a přiblíží podrobnosti o návrhu, implementaci a testování uživatelského scénáře.

7.1 Stav podpory archivovaných feedů před implementací

Editor umožňoval jako umístění vstupního feedu přijmout URL souboru, který byl ve formátu XML, ZIP nebo GZIP. Některé feedy byly komprimovány do archivů především za účelem snížení velikosti souboru. Všechny tyto soubory se při zpracování stáhly a jejich obsah se dále nekontroloval. V případě, že byl stažený feed archivován, aplikace se pokusila při zpracování feedu tento soubor číst v závislosti na detekovaném MIME type souboru. Pokud byl detekován MIME type *application/x-gzip*, pokusila se aplikace číst pomocí zabudovaného compression stream API *compress.zlib://* [25]. V případě, že MIME type odpovídal *application/zip*, soubor byl čten pomocí zabudované knihovny *ZipArchive* [26].

Problém nastal v případě, že stažený archiv s feedem nebyl v očekávaném stavu. Mohlo se stát, že stažený archiv nebyl kompletní, a proto čtení archivu selhalo. Mohla také nastat situace, při které byl archiv prázdný, nebo neobsahoval XML feed. Možným řešením tohoto uživatelského scénáře bylo odhalení nepoužitelných archivů ihned po stažení. Cílem bylo zajistit plynulé generování feedu s případným využitím předešlé čitelné verze feedu. Řešení také mělo obsáhnout stejnou kontrolu i při registraci nového feedu do systému s účelem zjednodušení tvorby feedu uživatele.

7.2 Návrh implementace kontroly archivů

Výhodou současné implementace je, že metoda pro získání cesty souboru včetně wrapperu je oddělena do samostatné služby, kterou lze použít kdekoliv v modelu. Celá tato funkcionalita tedy může být bezpečně upravena na jednom místě. Prvním krokem je doplnění této služby o pokus o otevření archivu. V případě, že tento pokus selže, můžeme archiv považovat za nevhodný pro další práci z důvodu nedostupného obsahu. Pokud archiv otevřít můžeme, následuje kontrola jeho obsahu. Služba vyhledá první položku archivu, kterou považuje za chtěný XML feed. Pokud metoda pro vyhledání první položky nezjistí název souboru, znamená to, že se jedná o prázdný archiv a není tedy z hlediska aplikace nijak použitelný. Pro oba neúspěšné scénáře vyvolá služba výjimku o nečitelném archivu, kterou je nutné dále zpracovat v závislosti na původním požadavku o čtení archivu.

7.3 Implementace kontroly archivovaných feedů

První variantou je archivovaný feed ve formátu GZIP. Při využití zabudovaného file streamu *zlib://* umožňuje PHP s takovýmto souborem pracovat jako s jakýmkoliv jiným běžným souborem a pro jeho čtení lze použít vestavěné funkce pro práci se soubory. V případě, že je požadována cesta k souboru ve formátu přijatelném jako argument pro funkci pro otevření souboru (např. *fopen*), výsledkem je řetězec složený z tohoto wrapperu a názvu souboru. Následná kontrola obsahu je provedena v komponentách využívajících tuto službu v závislosti na požadovaném obsahu souboru.

Druhou variantou je archivovaný feed ve formátu ZIP. Pro čtení těchto souborů využijeme zmíněnou knihovnu *ZipArchive*. Kontrola pomocí této knihovny je snazší, jako první probíhá pokus o otevření archivu, pokud selže, je vyvolána výjimka obsahující chybový kód vrácený při otevření archivu. V případě, že otevření proběhlo v pořádku, načte knihovna název prvního souboru v archivu. Pokud toto načtení selže, je opět vyvolána výjimka. Pokud otevření archivu i načtení první položky proběhly v pořádku, je výsledkem řetězec složený z wrapperu *zip://* a názvu prvního souboru. Případná další kontrola obsahu je opět delegována na stranu využívající tuto službu.

Pro nevyhovující stavy archivu vázané k této službě je vytvořena *Výjimka pro filestream wrapper*. Výjimka je navržena dle standardu aplikace, a to s využitím návrhového vzoru *Static factory methods* (někdy označováného jako Named constructor). První výhodou takového návrhu při vytváření výjimek je přidaný kontext samotného vyvolání výjimky. Místo pouhého vytvoření výjimky použitím *new* vytváříme instanci za pomoci metody konkrétně popisující příčinu selhání a prostředků nevyhovujícím systému (metoda fromUnopenableZipArchiveFilePathAndErrorCode). Druhou výhodou je delegování tvorby chybové zprávy výjimky samotné výjimce, což umožňuje konzistentní zprávy skrze všechna vyvolání této výjimky a obecně příjemnější práci s výjimkou.

Tato třída rozšiřuje nativní výjimku PHP *DomainException* a implementuje rozhraní *HttpExceptionInterface*. Toto rozhraní je v systému využité pro výjimky, které jsou odchyceny až na úrovni middleware aplikace. Výjimky implementující toto rozhraní, které nejsou odchyceny, pozmění http kód odpovědi na kód uvedený v takové výjimce. Toto slouží pro korektní signalizaci http kódy při využití koncových bodů aplikace.

```
<?php declare(strict_types=1);

namespace \Core\Components\\Parser\Exception;

use \Client\Api\IResponse;
use \Core\Components\Http\HttpExceptionInterface;
```

```
class FileStreamWrapperException extends \DomainException implements
    HttpExceptionInterface {

   private function __construct(string $message) {
      parent::__construct($message);
   }

   public static function fromUnopenableZipArchiveFilePathAndErrorCode(string
       $filePath, int $errorCode): self {
      return new self(sprintf('Cannot open zip archive at "%s" (ZipArchive
          error code: %d)', $filePath, $errorCode));
   }

   public static function fromEmptyZipArchiveFilePath(string $filePath): self {
      return new self(sprintf('Cannot find any file in zip archive at "%s"',
          $filePath));
   }

   /**
    * @return int
    */
   public function getHttpCode() {
      return IResponse::HTTP_CODE_UNPROCESSABLE_ENTITY;
   }
}
```

Výpis 3: Definice výjimky pro filestream wrapper

7.4 Revize práce s archivovanými feedy

Funkcionalita je zpracována v rámci jednoho uživatelského scénáře. Celkový proces začal návrhem scénáře z původní chyby v produkčním prostředí a končí nahráním této úpravy na produkční server. Nasazením této úpravy do produkčního prostředí je zajištěna lepší informovanost klientů o důvodu odmítnutí jejich feedů systémem, jelikož dosavadní řešení poskytovalo pouze obecnou chybu. Tato funkcionalita je použita jak pro periodické zpracovaní feedu, tak pro registraci nového feedu. Celé řešení pokrývají Unit testy, které pro kontrolu využívají prázdné i poškozené archívy.

8 Závěr

Závěrem hodnotím praxi rozhodně kladně už jen z důvodu, kolik jsem se toho za tak krátkou dobu naučil. Byly mi konkrétně představeny koncepty, které byly známy díky studiu, ale dosud nebyly nijak prakticky využity. Také jsem získal podrobné povědomí o tom, jak široce se může vývojář angažovat při vývoji vlastního nástroje určeného pro specifickou klientelu. V rámci praxe jsem byl seznámen s e-commerce v Česku a střední Evropě. Absolvování individuální odborné praxe ve firmě osobně vnímám jako velmi důležitou zkušenost a skvělou příležitost pro start kariérního rozvoje.

8.1 Znalosti získané studiem, které byly uplatněny v praxi

Přínos studia a získaných znalostí studiem využitých v praxi rozhodně nelze popřít. Vždy jsem za velmi přínosné považoval předměty *Algoritmy I* i *Algoritmy II* a jejich přidaná hodnota se ukázala i při výkonu této praxe, jelikož dobře obecně rozvíjejí logické a analytické myšlení.

Dalším zdrojem využitých znalostí je předmět *Vývoj informačních systémů*, jehož praktická náplň se nejvíce podobala úkonům prováděným v této praxi. Součástí každého úkolu byla analýza a dekompozice složitého úkolu na dílčí celky, jeho návrh, implementace a testování.

Předměty *Úvod do databázových systémů* a *Databázové a informační systémy* poskytly dobrý základ pro práci s databázovou vrstvou i přes fakt, že ani jeden nepředstavil úložiště využité v rámci zpracovávaného projektu. Za zmínku hlavně stojí práce a tvorba vlastního objektově-relačního mapperu, který byl v praxi použit ve formě PHP frameworku Doctrine ORM. Ten názorně představil techniky použité pro efektivní komunikaci aplikace a databázové vrstvy.

Poslední využité znalosti, které uvedu, pochází především z předmětů *Programovací jazyky I*, *Programovací jazyky II*, *Programování I* a *Programování II*. Jedná se o schopnost programovat nezávisle na použitém jazyce podle dobrých logických úvah, obecného návrhu, standardů a konvencí. Jazyk PHP prakticky představen v rámci studia nebyl, je však natolik podobný jazykům, které mi představeny byly, že využití tohoto jazyka nepůsobilo téměř žádné obtíže.

8.2 Chybějící znalosti získané praxí

Jako první scházející znalost, která nebyla předmětem studia, je jazyk PHP. Nejednalo se však o příliš velký problém, protože jsem se v rámci studia setkal jazyky a frameworky, jimiž se PHP i použité frameworky inspirovaly. Příkladem technologie, která byla v rámci studia představena pouze minimálně, je Git. V rámci praxe byla má znalost Gitu značně prohloubena. Další neméně důležitou problematikou, která při studiu nebyla ukázána v praxi je testování. Testování zastává velmi důležitou úlohu při vývoji a jeho představení v rámci studia značně pomůže studentům při uplatnění v praxi. Poslední znalostí, kterou bych při studiu ocenil je praktická ukázka agilních metodik vývoje. Vzhledem k struktuře studia se samo nabízí praktické využití agilních metodik a iterativního vývoje, například při tvorbě projektů.

Literatura

1. *Biddingtools s.r.o.* [online] [cit. 2020-05-01]. Dostupné z: https://biddingtools.cz/.
2. *ette capital a.s.* [online] [cit. 2020-05-01]. Dostupné z: https://www.ettecapital.com/o-nas/.
3. LOPEZ, Antonio. *Learning PHP 7*. Packt Publishing, 2016.
4. *Nette* [online] [cit. 2020-05-01]. Dostupné z: https://nette.org/cs/.
5. *Slim Framework* [online] [cit. 2020-05-01]. Dostupné z: http://www.slimframework.com/.
6. *MariaDB* [online] [cit. 2020-05-01]. Dostupné z: https://mariadb.org/.
7. *Doctrine* [online] [cit. 2020-05-01]. Dostupné z: https://www.doctrine-project.org/.
8. EVANS, Eric. *Domain-Driven Design : Tackling Complexity in the Heart of Software*. Pearson Education (US), 2003.
9. T. RAY, Erik. *Learning XML, Second Edition*. O'Reilly, 2001.
10. POTENCIER, Fabien. *Symfony 5: The Fast Track*. Symfony SAS, 2019.
11. *Guzzle* [online] [cit. 2020-05-01]. Dostupné z: http://docs.guzzlephp.org/en/stable/.
12. *Swift Mailer* [online] [cit. 2020-05-01]. Dostupné z: https://swiftmailer.symfony.com/.
13. *Twig* [online] [cit. 2020-05-01]. Dostupné z: https://twig.symfony.com/.
14. *git* [online] [cit. 2020-05-01]. Dostupné z: https://git-scm.com/.
15. *PHPUnit* [online] [cit. 2020-05-01]. Dostupné z: https://phpunit.de/.
16. *PHPStan* [online] [cit. 2020-05-01]. Dostupné z: https://phpstan.org/user-guide/getting-started.
17. *PHP Code Sniffer* [online] [cit. 2020-05-01]. Dostupné z: https://github.com/squizlabs/PHP_CodeSniffer.
18. *PHP Mess Detector* [online] [cit. 2020-05-01]. Dostupné z: https://phpmd.org/.
19. *Bitbucket Cloud* [online] [cit. 2020-05-01]. Dostupné z: https://bitbucket.org.
20. *Circle CI* [online] [cit. 2020-05-01]. Dostupné z: https://circleci.com/.
21. *Heureka feed* [online] [cit. 2020-05-01]. Dostupné z: https://sluzby.heureka.cz/napoveda/xml-feed/.
22. *Google Shopping feed* [online] [cit. 2020-05-01]. Dostupné z: https://support.google.com/merchants/answer/7052112?hl=cs.
23. *PHP DOM* [online] [cit. 2020-05-01]. Dostupné z: https://www.php.net/manual/en/book.dom.php.

24. *SOLID principle* [online] [cit. 2020-05-01]. Dostupné z: https://medium.com/@dhkelmendi/solid-principles-made-easy-67b1246bcdf.

25. *PHP compress streams* [online] [cit. 2020-05-01]. Dostupné z: https://www.php.net/manual/en/wrappers.compression.php.

26. *PHP ZipArchive* [online] [cit. 2020-05-01]. Dostupné z: https://www.php.net/manual/en/class.ziparchive.php.